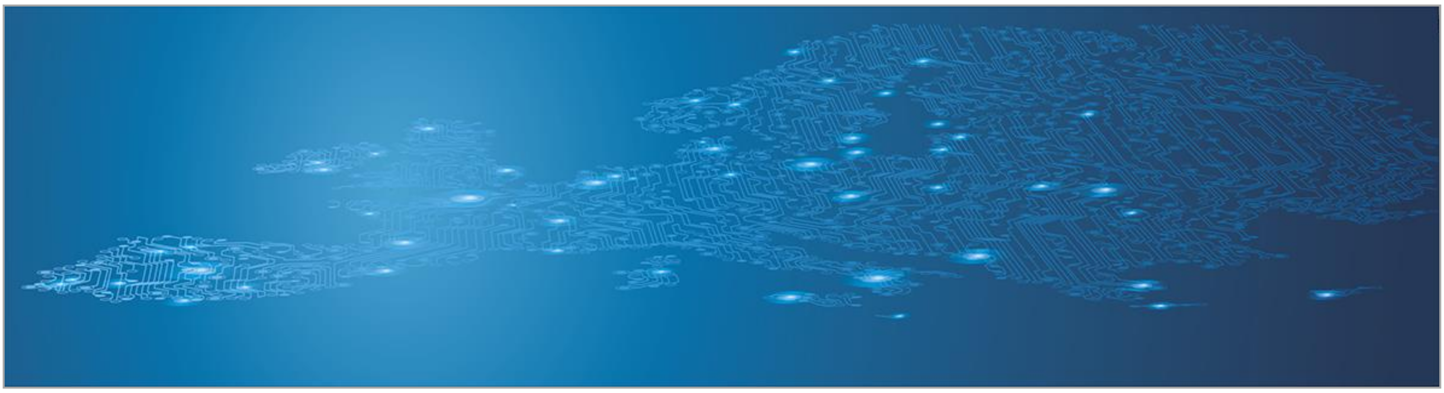

Developing countermeasures (signatures, indicators of compromise)

Toolset, Document for students

December 2014

European Union Agency for Network and Information Security www.enisa.europa.eu

About ENISA

The European Union Agency for Network and Information Security (ENISA) is a centre of network and information security expertise for the EU, its member states, the private sector and Europe's citizens. ENISA works with these groups to develop advice and recommendations on good practice in information security. It assists EU member states in implementing relevant EU legislation and works to improve the resilience of Europe's critical information infrastructure and networks. ENISA seeks to enhance existing expertise in EU member states by supporting the development of cross-border communities committed to improving network and information security throughout the EU. More information about ENISA and its work can be found at www.enisa.europa.eu.

Authors

This document was created by Lauri Palkmets, Cosmin Ciobanu, Yonas Leguesse, and Christos Sidiropoulos in consultation with DFN-CERT Services[1] (Germany), ComCERT[2] (Poland), and S-CURE[3] (The Netherlands).

Contact

For contacting the authors please use cert-relations@enisa.europa.eu

For media enquires about this paper, please use press@enisa.europa.eu.

Acknowledgements

ENISA wants to thank all institutions and persons who contributed to this document.

[1] Klaus Möller, and Mirko Wollenberg

[2] Mirosław Maj, Tomasz Chlebowski, Krystian Kochanowski, Dawid Osojca, Paweł Weżgowiec, and Adam Ziaja

[3] Michael Potter, Alan Robinson, and Don Stikvoort

Legal notice

Notice must be taken that this publication represents the views and interpretations of the authors and editors, unless stated otherwise. This publication should not be construed to be a legal action of ENISA or the ENISA bodies unless adopted pursuant to the Regulation (EU) No 526/2013. This publication does not necessarily represent state-of the-art and ENISA may update it from time to time.

Third-party sources are quoted as appropriate. ENISA is not responsible for the content of the external sources including external websites referenced in this publication.

This publication is intended for information purposes only. It must be accessible free of charge. Neither ENISA nor any person acting on its behalf is responsible for the use that might be made of the information contained in this publication.

Copyright Notice

Table of Contents

Main Objective	In this exercise the students will learn how to leverage information gathered during analysis into actionable signatures. Both network and system oriented signatures will be discussed.	
Targeted Audience	CERT Technical specialists. The exercise will use information gathered during previous exercises 'Artifact analysis fundamentals' and 'Advanced artifact analysis', these are likewise recommended as prerequisites.	
Total Duration	Approx. 8.0 hours	
Time Schedule	Introduction to Snort rules, and Yara patterns.	3 hours
	Task 1: Developing Snort rules	2.0 hours
	Task 2: Developing Yara patterns	2.0 hours
	Summary of the exercise	0.5 hour
Frequency	Once per team	

1 Objective and Description

The exercise begins with an introduction to Yara and Snort signature creation. Additionally, the exercise covers signature syntax, descriptions of methods, how to make best use of different options, and the main differences between the two tools.

Further, students will create Yara and Snort signatures, based on a set of results of malware analysis conducted in previous exercises. After the creation of signatures, verification is performed. Yara signatures are checked by analysing the files, and performing a verification to see if the samples belong to the same family of malware samples identified (no false positive hits). Snort signatures will be verified based on the set of network traffic capture (PCAP) files prepared earlier. Similar to the Yara, students should look into capture files and identify suspicious traffic, and avoid false positive hits.

Students will learn how to leverage on information gathered during analysis into actionable signatures. Both network and system oriented signatures will be discussed.

The training is intended for CERT technical specialists. This exercise will use information gathered during analysis conducted in the previous exercises.

2 General description

The goal of this exercise is to enable students to use information gathered during malware analysis for the purpose of identifying compromised systems using automated tools. To accomplish this, two approaches have been chosen to describe identification patterns of malware behaviour. Both are open source, they are implemented in various tools, and they are used in the wild. One of the tools (Yara) focuses on system evidence. The other (Snort[4]) focuses on patterns found in network traffic.

The information used in the signatures is derived from analysis in previous exercises.

This exercise starts with an introduction to the two formats, and will provide some background information like tools which use the formats to identify compromised systems or alternative approaches to achieve the same goal.

After the introduction, each format will be handled in a separate task. The trainer will provide one example to convert analysis information into an actionable pattern in each task. Afterward, the students will use the information gathered from the previous exercises to write signatures and test them in a hands-on setting.

3 Exercise Course

3.1 Introduction

Developing malware signatures from information gathered during the analysis step is an important part of the incident response process as it defines the line between detection and reaction/correction. Being able to transform identified characteristics of malware behaviour (both system and network related) into signatures and patterns, which can be used by off the shelf software to identify compromised systems, supports an organisation's recovery from an incident.

[4] Open source network intrusion prevention and detection system http://snort.org/

4 Developing Snort signatures

4.1 Introduction

There will be three mandatory parts for this exercise and some optional add-ons. The first example will be guided by the trainer to demonstrate the process and to provide the students with a hands-on example. There are two different tasks for the students, one provided with information gathered during the analysis in the previous exercise and one based on information gathered by network based analysis tools (MITMProxy, Tcpdump, and Wireshark).[5]

The necessary information for all three tasks is placed in the corresponding subdirectory of the training material. This is to provide the trainer with the possibility of starting the exercise with a clean sheet for all students or to be able to use the tasks without prerequisite exercises.

In all tasks Snort is used to test the signatures.

4.2 Snort syntax

The Snort website provides a thorough documentation of the rules syntax.[6] Here we will focus on the basic structures and explain the main parts of the approach in the trainers example walk-through.

Parts of a rule: **Rule headers, Rule options**

Parts of the rule headers: **Action(s)**, **Protocol(s)**, **IP Address(es)**, **Port(s)**, **Direction(s)**

Parts of the rule options: **General**, **Payload**, **Non-Payload**, **Post-Detection**

4.2.1 Keywords

There are keywords which are often used to define and narrow down the length of rules. The most important are documented below and more can be found in the official Snort documentation:[7]

Flow
Defines the direction and state of the traffic on which the rule will be activated.

Content
Contains a pattern that is searched for in the packet payload. It can be manipulated by further keywords following in the rule options.

Nocase
Deactivates case matching for the previous 'content' keyword in the rule.

Offset
Marks the position in the packet to start searching for the pattern defined in the previous 'content' keyword.

Depth
Defines how far an IDS should search for a pattern in a packet as defined in the previous 'content' declaration.

---

[5] Tools are covered in exercises 1 and 2 of this set.
[6] Writing Snort Rules http://manual.snort.org/node27.html
[7] Payload Detection http://manual.snort.org/node32.html

Pcre
This keyword can be used to write patterns in regular expressions.

Classtype
Classtype contains a single or combined word to classify the type of event which has triggered the rule.

Sid
Each rule is identified by a unique Snort rule identifier (sid). Sid's above 1.000.000 can be used for local rules.

Msg
This option contains a description of the event which will be logged and gives an analyst an impression regarding the nature of an incident.

Reference
Rule writers can include links and pointers to vulnerability databases (CVE, OSVDB, general URL).

4.2.2 Perl Compatible Regular Expressions (PCRE) excursion

The Perl Compatible Regular Expressions library provides a set of functions as an API to enable applications to use the Perl syntax to define regular expressions. The usage of this library allows the snort operator to define very flexible matching rules. For example, the following rule tries to match Kelihos download activity and uses PCRE to match the binary names of a certain set of malware samples:

Example:

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN
Possible Kelihos.F EXE Download Common Structure 2";
flow:to_server,established; content:"/mod"; depth:4; nocase;
http_uri; content:".exe"; nocase; http_uri; fast_pattern:only;
pcre:"/^\/mod[12]\/[^\/]+?\.exe$/Ui"; content:!"User-Agent|3a|";
http_header; nocase; content:"Host|3a|"; depth:5; http_header;
reference:md5,9db28205c8dd40efcf7f61e155a96de5; classtype:trojan-
activity; sid:2018395; rev:3;)
```

The following is an explanation of the PCRE syntax of this example:

```
pcre:"/^\/mod[12]\/[^\/]+?\.exe$/Ui"
```

The expression itself is contained between / markings, followed by post-expression modifiers U and i. The latter tells Snort to match the expression without regard to case and the former to match the decoded URI.

The ^ marks the beginning of the URI string. The backslash "escapes" the following slash, meaning to use a literal / followed by the string mod and the digit 1 or 2. Afterwards we have another escape slash. The expression in the squared bracket modified by +? means all further slashes and the content in between will be matched exactly.
The term \.exe$ signifies the string ".exe" being the end of the line.

5 Students task 1

The students will analyse a Ramnit[8] sample. Following the information will be presented to the trainer.

Cuckoo Sandbox report can be used along with PCAP file (/home/enisa/enisa/ex5/malware/ramnit/).

Trainees should sort the collected data and comment where feasible.

Hostname	IP	Comment
awrcaverybrstuktdybstr.com	66.228.49.83	HTTPS connection
google.com	74.125.227.200 74.125.227.197 74.125.227.193 74.125.227.199 74.125.227.206 74.125.227.192 74.125.227.201 74.125.227.194 74.125.227.198 74.125.227.195 74.125.227.196	Benign, possibly used to check connectivity
awecerybtuitbyatr.com	66.228.49.83	HTTPS connection

There is only sparse network related information available. We have two odd host names, which are directly related to the malware function, but the traffic itself is SSL encrypted. If data gathered by MITMProxy is available, this would enhance the analysis but not necessarily improve the rule's quality. So students are left with using the following option:

DNS requests to one or both of awrcaverybrstuktdybstr.com and awecerybtuitbyatr.com domains.

Rule header: alert **`udp $HOME_NET any -> $EXTERNAL_NET 53`**

The payload matching the hexadecimal presentation has been chosen as it is better in resource efficiency (no translation from ASCII by Snort) and more accurate as there would be no encoding errors:

```
(msg:"ENISA EXERCISE outgoing ramnit DNS request"; classtype:trojan-activity; content:"|11 61 77 65 63 65 72 79 62 74 75 69 74 62 79 61 74 72 03 63 6f 6d 00 00 01 00 01|"; sid:10000010;)
```

Change to the directory: /home/enisa/enisa/ex5 to test the rule.

Invoke rule2alert like this:

```
python addons/rule2alert-read-only/r2a.py -v -c snort/snort.test.conf -m 192.168.0.0/16 -e 192.0.2.53/32 -f snort/enisa-snort-rule-2.rules -w snort/enisa-exercise-test2.pcap
```

Check the file with Wireshark.

[8]Ramnit Goes Social http://www.seculert.com/blog/2012/01/ramnit-goes-social.html

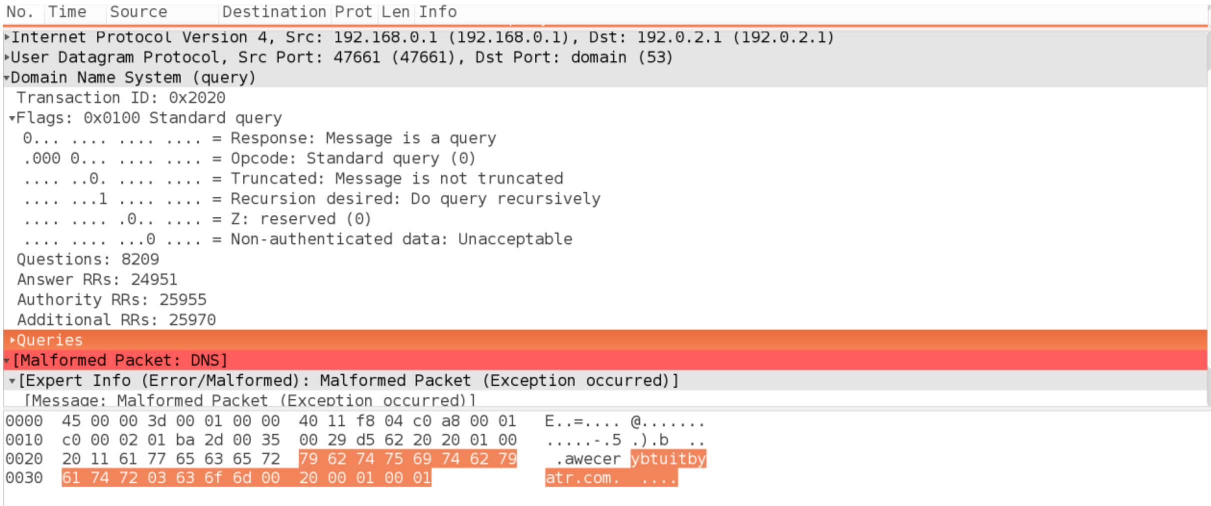

Figure 1: Student task 1 Wireshark screenshot

Note the warning regarding a malformed DNS packet, after that invoke Snort and let it read the PCAP you created:

```
snort -d -c snort/snort.test.conf -q -A console -k none -r snort/enisa-exercise-test-2.pcap
```

You should see the following output:

```
08/20-11:42:19.673960 [**] [1:10000010:0] ENISA EXERCISE outgoing ramnit DNS query [**] [Classification: A Network Trojan was Detected] [Priority: 1] {UDP} 192.168.0.1:21837 -> 192.0.2.1:53
```

The created rule matches the traffic; nevertheless, further refinement for efficiency and protocol comprehension is recommended.

```
Content: ”|01 00 00 01 00 00 00 00 00 00|”; offset:2;
```

The hex content signifies a recursive DNS query. Offset tells Snort to start matching the payload 2 bytes after the start of the packet payload.

```
Distance:0; content:"|00 01 00 01|"; distance:0;
```

Distance:0 lets Snort match the pattern only if directly after the previous match the following hex code 00 01 00 01 matches.

The complete rule is presented as follows.

```
alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg:"ENISA EXERCISE outgoing ramnit DNS query"; classtype:trojan-activity; content:"|01 00 00 01 00 00 00 00 00 00|"; offset:2; content:"|11 61 77 65 63 65 72 79 62 74 75 69 74 62 79 61 74 72 03 63 6f 6d 00|"; distance:0; content:"|00 01 00 01|"; distance:0; sid:10000011;)
```

6 Students task 2

In this task, the Cuckoo report will not contain a clear indication of network activity. The following solution is based on information collected by running the UNIX tool 'strings' on the malware binary.

There is only actionable information in the 'strings' output.

`cd /home/enisa/enisa/ex5/malware/poisonivy/ && strings -a malware-poisonivy.exe`

The '-a' switch forces a scan of the whole file instead of initialized sections. This is of course only necessary when, e.g. scanning an ELF binary on Linux, but is mentioned here for completeness.

```
126,13,0$ strings -a malware-pisonivy.exe > strings-poisonivy.txt
127,13,0$ less strings-poisonivy.txt
```

Figure 2: Student task 2 strings command

During the analysis of the strings output an interesting host name can be detected.

```
SOFTWARE\Classes\http\shell\open\commandV
Software\Microsoft\Active Setup\Installed Components\
thecrusher
thecrusher.no-ip.biz
admin
msnpro
{04AC5F42-0A94-7D2E-A7BE-A4BA277243CF}
)!VoqA.I4
PPPPPP
WPPP
PPPP
SOFTWARE\Microsoft\Windows\CurrentVersion\Run
SOFTWARE\Microsoft\Windows\CurrentVersion\Run
explorer.exe
```

Figure 3: Students task 2 strings output

In this case, this is the only actionable item to be found is domain name, so this is used in order to create a rule.

Hostname	IP	Comment
thecrusher.no-ip.biz	n/a, dynamic	no-ip provides dynamic DNS services under the domain no-ip.biz

`alert udp $HOME_NET any -> $EXTERNAL_NET 53`

For this step, it is recommended to convert the string 'thecrusher.no-ip.biz' into hexadecimal[9] as it increases resource efficiency (no translation from ASCII by Snort) and is more accurate as it avoids encoding errors:

`thecrusher.no-ip.biz - 74 68 65 63 72 75 73 68 65 72 2e 6e 6f 2d 69 70 2e 62 69 7a`

Example: **`(msg:"ENISA EXERCISE outgoing Poison Ivy DNS request"; classtype:trojan-activity; content:"|74 68 65 63 72 75 73 68 65 72 2e 6e 6f 2d 69 70 2e 62 69 7a|"; sid:10000020;)`**

Navigate to the directory: /home/enisa/enisa/ex5 and invoke rule2alert like this.

[9]Hex To ASCII Converter http://dolcevie.com/js/converter.html

```
python addons/rule2alert-read-only/r2a.py -v -c snort/snort.test.conf -m 192.168.0.0/16 -e 192.0.2.53/32 -f snort/enisa-snort-rule-3.rules -w snort/enisa-exercise-test3.pcap
```

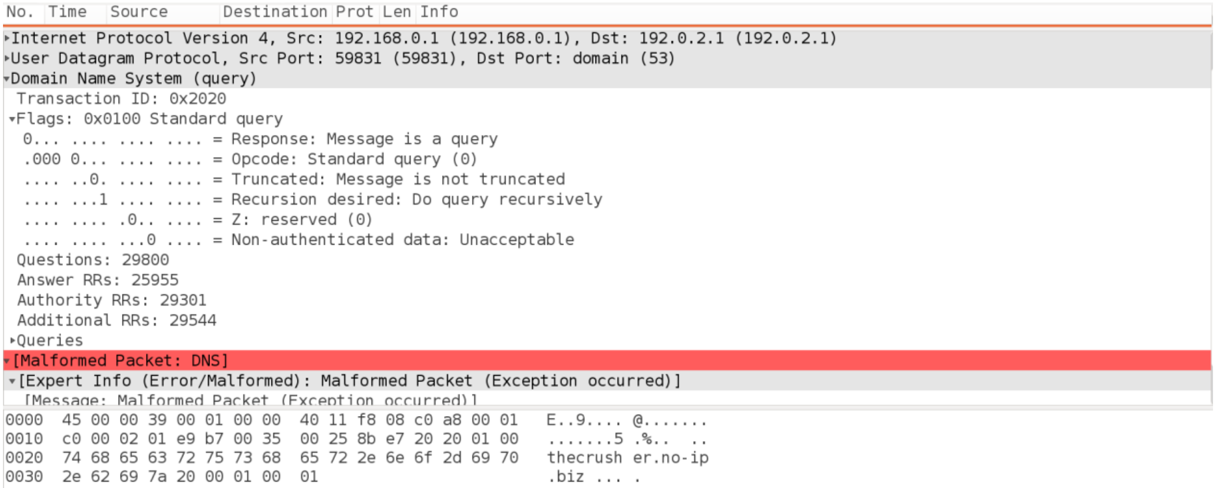

Figure 4: Student task 2 Wireshark screenshot

Review the file with Wireshark application and note the warning regarding malformed DNS packet.

Invoke Snort and let it read your created PCAP: **`snort -d -c snort/snort.test.conf -q -A console -k none -r snort/enisa-exercise-test-3.pcap`**

You should see the following output: **`08/22-10:51:37.672281 [**] [1:10000020:0] ENISA EXERCISE outgoing Poison Ivy DNS query [**] [Classification: A Network Trojan was Detected] [Priority: 1] {UDP} 192.168.0.1:57192 -> 192.0.2.1:53`**

The created rule matches the traffic, nevertheless a refinement in terms of efficiency and protocol comprehension is recommended.

Example: **`alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg:"ENISA EXERCISE outgoing Poison Ivy DNS query"; classtype:trojan-activity; content:"|01 00 00 01 00 00 00 00 00 00|"; offset:2; content:"|74 68 65 63 72 75 73 68 65 72 2e 6e 6f 2d 69 70 2e 62 69 7a|"; distance:0; content:"|00 01 00 01|"; distance:0; sid:10000021;)`**

7 Developing Yara patterns

In this task description we use excerpts from the official Yara documentation (http://yara.readthedocs.org/en/latest/index.html).

7.1 Yara

Yara is a tool aimed at but not limited to helping malware researchers to identify and classify malware samples. With Yara descriptions of malware families can be created based on textual or binary patterns. Each description or rule consists of a set of strings and a boolean expression which determines its logic.

Yara was installed during the '*Building artifact handling and analysis environment*' exercise as one of the Cuckoo sandbox dependencies. For this exercise, create the directory *yara* in /home/enisa/.

```
$ mkdir /home/enisa/yara
$ cd /home/enisa/yara
```

7.2 Developing Yara patterns[10]

Yara rules are easy to write and understand, and they have a syntax that resembles the C language.

Example Yara rule:

```
rule ExampleRule
{
  strings:
    $my_text_string = "text here" /* Text strings are enclosed on double quotes just like in the C language */
    $my_hex_string = { E2 34 A1 C8 23 FB } /* Hex strings are enclosed by curly brackets, and they are composed by a sequence of hexadecimal numbers that can appear contiguously or separated by spaces */
    $my_regexp = /md5: [0-9a-zA-Z]{32}/ /* Regular expressions are defined in the same way as text strings, but enclosed in backslashes instead of double-quotes, like in the Perl programming language */

  condition:
    $my_text_string or $my_hex_string or $my_regexp
}
```

Each rule in Yara starts with the keyword rule followed by a rule identifier – in the above example the identifier is "ExampleRule".

Rules are generally composed of two sections: strings definition and condition. The strings definition section can be omitted if the rule doesn't rely on any string, but the condition section is required. Decimal numbers are not allowed in hex strings. You can add comments to your YARA rules just as if it was a C source file, both single-line and multi-line C-style comments are supported. Conditions are nothing more than Boolean expressions as found in all programming languages.

Yara keywords: all, and, any, ascii, at, condition, contains, entrypoint, false, filesize, fullword, for, global, in, import, include, int8, int16, int32, matches, meta. nocase, not, or, of, private, rule, strings, them, true, uint8, uint16, uint32, wide.

---

[10]We use the introduction to developing Yara patterns from Victor M. Alvarez in the first paragraphs, the original text can be found in the official Yara documentation at: https://github.com/plusvic/yara/blob/master/docs/writingrules.rst

In this exercise we will use malware sample “aop.exe” from previous exercise. Create a directory called **malware** and copy the file “aop.exe” to **/home/enisa/yara/malware** directory:

```
$ cd /home/enisa/yara
$ mkdir malware
$ cp /home/enisa/enisa/ex5/malware/aop.exe malware/
```

At the beginning we will need to extract strings from this sample. To obtain the list of all strings under the Linux “strings” tool can be used.

```
$ strings malware/aop.exe | more
```

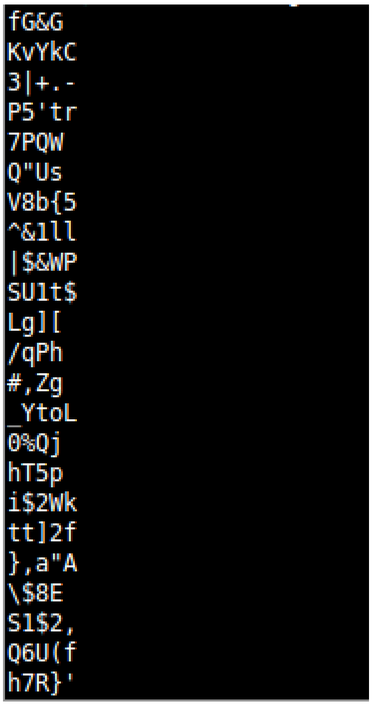

Figure 5: Strings found in aop.exe file

We will build the first simple rule, create a file called ‘enisa.yara’ using any text editor of your choice (we use nano in this example):

```
$ cd /home/enisa/yara
$ nano enisa.yara
```

```
rule ENISA
{
    strings:
        $foo1 = "fG&G"
        $foo2 = "KvYkC"
        $foo3 = "3|+.-"
        $foo4 = "P5'tr"
```

```
        $foo5 = "7PQW"

    condition:
        $foo1 and $foo2 and $foo3 and $foo4 and $foo5
}
```

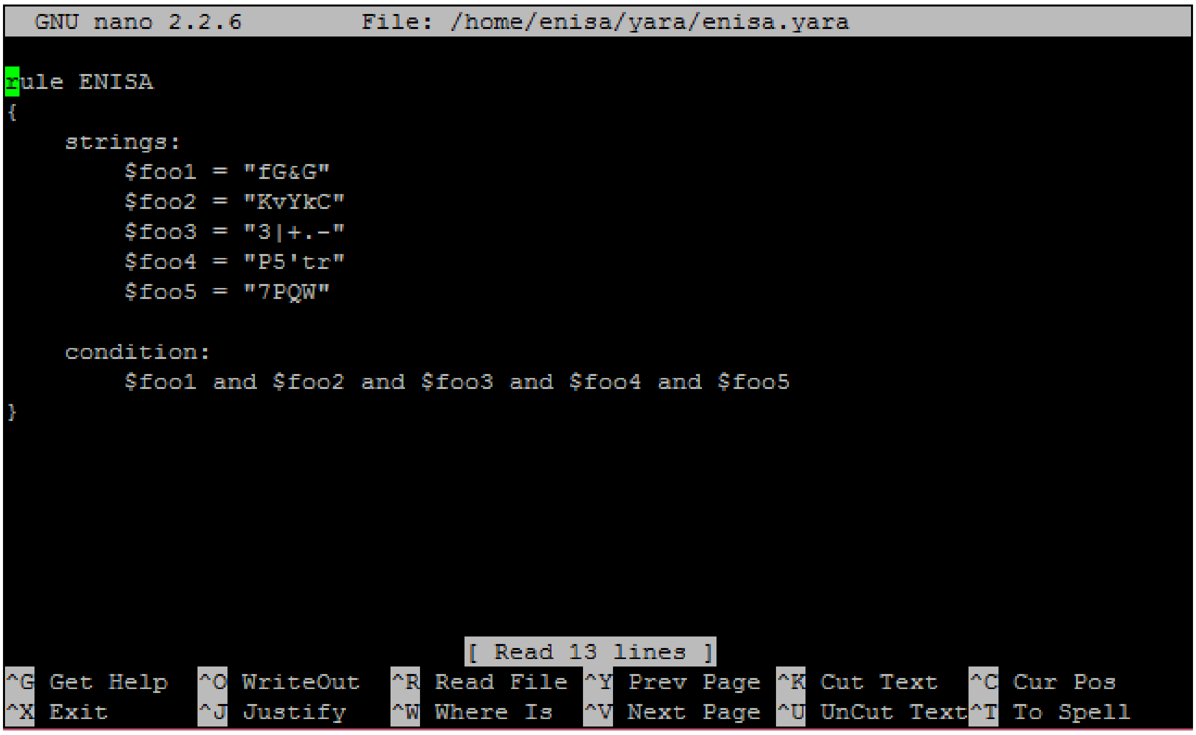

Figure 6: Editing /home/enisa/yara/enisa.yara file

(The file with that rule can be found in /home/enisa/enisa/ex5/rules/1.yara.)

Our rule will have the name “ENISA” and will be matched only when all the strings will occur in the file according to the conditions specified.

Our rule is done. Now we need to check for hits by typing the following commands in the console:

```
$ cd /home/enisa/yara/
$ yara enisa.yara malware/aop.exe
ENISA aop.exe
```

```
enisa@styx:~/yara$ yara enisa.yara malware/aop.exe
ENISA malware/aop.exe
enisa@styx:~/yara$ cat enisa.yara
rule ENISA
{
    strings:
        $foo1 = "fG&G"
        $foo2 = "KvYkC"
        $foo3 = "3|+.-"
        $foo4 = "P5'tr"
        $foo5 = "7PQW"

    condition:
        $foo1 and $foo2 and $foo3 and $foo4 and $foo5
}

enisa@styx:~/yara$
```

Figure 7: Patterns producing a hit in aop.exe examination

(The file with that rule can be found in /home/enisa/enisa/ex5/rules/2.yara.)

Output:

```
ENISA aop.exe
```

This output means that there is a hit in rule “ENISA” and file “aop.exe”. No output means that there is no hit.

We can also write the condition part in easier way such as *all of ($foo*)*:

```
rule ENISA
{
    strings:
        $foo1 = "fG&G"
        $foo2 = "KvYkC"
        $foo3 = "3|+.-"
        $foo4 = "P5'tr"
        $foo5 = "7PQW"

    condition:
        all of ($foo*)
}
```

(The file with that rule can be found in /home/enisa/enisa/ex5/rules/2.yara.)

This is equivalent to the previous rule. The difference is the ‘condition’ part where we replaced a logical conjunction of five named strings to be matched with a short construction requiring a match of all the strings defined in the section that begin with ‘foo’.

Beside the string definition and condition sections, rules can also have a metadata section where you can put additional information about your rule. The metadata section is defined with the keyword meta and contains identifier/value pairs:

```
rule ENISA
{
    meta:
        author = "ENISA"
        description = "malware"

    strings:
        $foo1 = "fG&G"
        $foo2 = "KvYkC"
        $foo3 = "3|+.-"
        $foo4 = "P5'tr"
        $foo5 = "7PQW"

    condition:
        all of ($foo*)
}
```

(The file with that rule can be found in /home/enisa/enisa/ex5/rules/3.yara)

Note that the identifier/value pairs defined in the metadata section cannot be used in the condition section. Their only purpose is to store additional information about the rule.

Our example malware is packed with UPX, we can do one single rule for both – packed and unpacked malware.

To make a copy and unpack malware type the following command in the console:

```
$ cd /home/enisa/yara/malware
$ cp aop.exe aop2.exe
$ sudo apt-get install upx
$ upx -d aop2.exe
```

```
enisa@styx:~/yara/malware$ cp aop.exe aop2.exe
enisa@styx:~/yara/malware$ upx -d aop2.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2011
UPX 3.08        Markus Oberhumer, Laszlo Molnar & John Reiser   Dec 12th 2011

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    135168 <-     52736   39.02%    win32/pe     aop2.exe

Unpacked 1 file.
enisa@styx:~/yara/malware$
```

Figure 8: Decompression of aop2.exe file

‘upx –d’ means decompress in the example above. Now we have packed the file “aop.exe” with UPX and unpacked “aop2.exe”.

To find common strings in both files, type the command:

```
$ comm -1 -2 <(strings aop.exe | sort) <(strings aop2.exe | sort)
```

```
enisa@styx:~/yara/malware$ comm -1 -2 <(strings aop.exe | sort) <(strings aop2.e
xe | sort)
~0;~,}
5866
7PQW
ADVAPI32.dll
<At;<Bt7
AVICAP32.dll
BitBlt
capCreateCaptureWindowA
ceil
DNQ
ExitProcess
FreeSid
GDI32.dll
GetDC
GetProcAddress
GFMu
ICOpen
InternetOpenUrlA
JGRW
KERNEL32.DLL
LoadLibraryA
M263
```

Figure 9: The strings common to both files

Command **comm -1 -2** shows what lines are in common in both strings while **<(strings aop.exe | sort)** returns a list of strings from “aop.exe”, then sorts it. Output sends as a string to compare.

Now we have list of strings that are in both binaries. As mentioned above, we can now build a single rule that matches both files.

```
$ cd /home/enisa/yara
```

```
$ nano enisa.yara
```

Replace content *enisa.yara* file with the following content:

```
rule ENISA
{
    strings:
        $ = "~0;~,}"
        $ = "5866"
        $ = "7PQW"
        $ = "<At;<Bt7"
        $ = "M263"
        $ = "m3W gP"
        $ = "n ux"
        $ = "U&OR"
        $ = "?_Xran@std@@YAXXZ"

    condition:
        all of them
}
```

(The file with that rule can be found in /home/enisa/enisa/ex5/rules/4.yara)

As we are not referencing any string individually, we do not need to provide a unique identifier for each of them. In those situations, you can declare anonymous strings with identifiers consisting only in the $ character.

Now we can test the rule by typing the following command in the console:

```
$ yara -r enisa.yara malware
```

Note the '-r' option conducts recursive search of the directories.

```
enisa@styx:~/yara/malware$ ls
aop2.exe  aop.exe
enisa@styx:~/yara/malware$ cd ..
enisa@styx:~/yara$ yara -r enisa.yara malware
ENISA malware/aop.exe
ENISA malware/aop2.exe
enisa@styx:~/yara$
```

Figure 10: Testing the rule shows two hits

Unpacked malware has more unique character strings. For example, we can find strings like prsionaljrq, prsionyta and providesmid.

Such unique names like "prsionaljrq, prsionyta and providesmid" usually distinctly identify a particular malware family. We can write rules which may detect new versions of this malware.

```
rule ENISA
{
    strings:
        $ = /prsionaljrq/i
        $ = /prsionyta/i
        $ = /providesmid/i

    condition:
        any of them
}
```

(The file with that rule can be found in /home/enisa/enisa/ex5/rules/5.yara)

We use simple regular expressions for case insensitive ("i" char after end of regexp – after "/") strings.

But this rule can generate false positives which will match, for example, an HTML file with saved news about this malware. To prevent this we add hex values:

```
rule ENISA
{
    strings:
        $mz = { 4d 5a } /* DOS header */
        $dos = { 54 68 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 6d 6f 64 65 } /* DOS stub */
        $s = /prsionaljrq/i
        $s = /prsionyta/i
        $s = /providesmid/i

    condition:
        $mz and $dos and any of ($s*)
}
```

(The file with that rule can be found in /home/enisa/enisa/ex5/rules/6.yara)

The above values were obtained by the command:

```
$ cd /home/enisa/yara/malware
$ hexdump -C aop2.exe | more
```

```
enisa@enisa:/tmp$ hexdump -C aop2.exe
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 08 01 00 00  |................|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000080  a1 87 8b 2e e5 e6 e5 7d  e5 e6 e5 7d e5 e6 e5 7d  |.......}...}...}|
00000090  d3 c0 ee 7d e7 e6 e5 7d  d3 c0 e1 7d e7 e6 e5 7d  |...}...}...}...}|
000000a0  9e fa e9 7d e7 e6 e5 7d  66 fa eb 7d e6 e6 e5 7d  |...}...}f..}...}|
000000b0  8a f9 ef 7d ee e6 e5 7d  8a f9 e1 7d e1 e6 e5 7d  |...}...}...}...}|
000000c0  e5 e6 e4 7d e6 e7 e5 7d  26 e9 b8 7d f2 e6 e5 7d  |...}...}&..}...}|
000000d0  0d f9 ef 7d e9 e6 e5 7d  0d f9 ee 7d f5 e6 e5 7d  |...}...}...}...}|
000000e0  22 e0 e3 7d e4 e6 e5 7d  52 69 63 68 e5 e6 e5 7d  |"..}...}Rich...}|
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000100  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 04 00  |........PE..L...|
00000110  ae 44 87 53 00 00 00 00  00 00 00 00 e0 00 0f 01  |.D.S............|
00000120  0b 01 06 00 00 60 01 00  00 a0 00 00 00 00 00 00  |.....`..........|
00000130  ec 54 01 00 00 10 00 00  00 70 01 00 00 00 40 00  |.T.......p....@.|
00000140  00 10 00 00 00 10 00 00  04 00 00 00 00 00 00 00  |................|
00000150  04 00 00 00 00 00 00 00  00 20 02 00 00 10 00 00  |......... ......|
00000160  00 00 00 00 02 00 00 00  00 00 10 00 00 10 00 00  |................|
```

Illustration 1: Hexadecimal dump of the executable file

```
00000000   4d 5a 90 00 03 00 00 00   04 00 00 00 ff ff 00 00
|MZ..............|
00000010   b8 00 00 00 00 00 00 00   40 00 00 00 00 00 00 00
|........@.......|
00000020   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
|................|
00000030   00 00 00 00 00 00 00 00   00 00 00 00 08 01 00 00
|................|
00000040   0e 1f ba 0e 00 b4 09 cd   21 b8 01 4c cd 21 54 68
|........!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f   |is
program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20   |t be
run in DOS |
00000070   6d 6f 64 65 2e 0d 0d 0a   24 00 00 00 00 00 00 00
|mode....$.......|
```

These values are characteristic for Windows binary files.

You can also create a less accurate rule using an automatic tool like YaraGenerator from https://github.com/Xen0ph0n/YaraGenerator. In this exercise, the *yaraGenerator.py* file is in the /home/enisa/enisa/ex5/ directory.

YaraGenerator depends on the python-pefile module. This module should be already installed as a result of the previous exercise. Otherwise you need to install it.

Copy the *yaraGenerator.py* script to */home/enisa/yara* and create a directory called *modules* with two files: *exe_blacklist.txt* and *exe_regexplacklist.txt*.

```
$ cd /home/enisa/yara
$ cp /home/enisa/enisa/ex5/yaraGenerator.py /home/enisa/yara
$ mkdir modules/ && touch modules/exe_blacklist.txt ; touch modules/exe_regexblacklist.txt
```

To generate the rule, type the following command:

```
$ python yaraGenerator.py -v -a ENISA -r ENISA -d malware -f exe malware/
```

```
enisa@styx:~/yara$ python yaraGenerator.py -v -a ENISA -r ENISA -d malware -f e
e malware/

[+] Generating Yara Rule ENISA from files located in: malware/

[+] Yara Rule Generated: ENISA.yar

  [+] Files Examined: ['7a0938b535f1bbd7a85065249bbbefd1', 'c2fbd09163178777376
c679c3bd8d34']
  [+] Author Credited: ENISA
  [+] Rule Description: malware
[+] Rule Below:

rule ENISA
{
meta:
        author = "ENISA"
        date = "2014-10-20"
        description = "malware"
        hash0 = "7a0938b535f1bbd7a85065249bbbefd1"
        hash1 = "c2fbd091631787773761c679c3bd8d34"
        sample_filetype = "exe"
        yaragenerator = "https://github.com/Xen0ph0n/YaraGenerator"
strings:
        $string0 = "OriginalFilename" wide
        $string1 = "LegalCopyright" wide
        $string2 = "yufan.com" wide
        $string3 = "1, 2, 0, 6" wide
        $string4 = "PrivateBuild" wide
        $string5 = "FileVersion" wide
        $string6 = "StringFileInfo" wide
        $string7 = "_Xran@std@@YAXXZ"
        $string8 = "<At;<Bt7"
        $string9 = "080404b0" wide
        $string10 = "VarFileInfo" wide
        $string11 = "1,=/4.1FA@6>D5H3>*@@;B;?>6@JI" wide
        $string12 = "VS_VERSION_INFO" wide
condition:
        12 of them
```

Illustration 2: Rule generated by the yaraGenerator

The settings used above are:

```
usage: yaraGenerator.py [-h] -r RULENAME [-a AUTHOR] [-d DESCRIPTION]
                          [-t TAGS] [-v] -f InputDirectory
```

```
YaraGenerator

positional arguments:
  InputDirectory        Path To Files To Create Yara Rule From

optional arguments:
  -h, --help            show this help message and exit
  -r RULENAME, --RuleName RULENAME
                        Enter A Rule/Alert Name (No Spaces + Must
Start with
                        Letter)
  -a AUTHOR, --Author AUTHOR
                        Enter Author Name
  -d DESCRIPTION, --Description DESCRIPTION
                        Provide a useful description of the Yara
Rule
  -t TAGS, --Tags TAGS  Apply Tags to Yara Rule For Easy Reference
                        (AlphaNumeric)
  -v, --Verbose         Print Finished Rule To Standard Out
  -f , --FileType         Select Sample Set FileType choices are:
unknown, exe,
                        pdf, email, office, js-html
```

8 Summary

This exercise focused on the technical aspects of converting actionable information found during the analysis of malware samples into rules and patterns, that can be deployed to intrusion detection systems (both network- and host-based).

The students learned how to dissect usable information for different pattern matching methods, and how to write simple signatures/rules. During the conclusion of the exercise, the trainer should focus on the process of collecting and sorting information, and identifying actionable information.

9 References

1. The pattern matching Swiss knife for malware researchers https://plusvic.github.io/yara/ (accessed 16. October 2014)
2. SNORT – Open source network intrusion prevention and detection system http://snort.org/ (accessed 16. October 2014)
3. SURICATA http://suricata-ids.org/ (accessed 16. October 2014)

4. The Bro Network Security Monitor https://www.bro.org/ (accessed 16. October 2014)
5. Bro Research Projects https://www.bro.org/research/index.html (accessed 16. October 2014)
6. A Practical Application of SIM/SEM/SIEM Automating Threat Identification https://www.sans.org/reading-room/whitepapers/logging/practical-application-sim-sem-siem-automating-threat-identification-1781 (accessed 16. October 2014)
7. Remotely Triggered Black Hole Filtering— Destination Based and Source Based http://web.archive.org/web/20060113035842/http://www.cisco.com/warp/public/732/Tech/security/docs/blackhole.pdf (accessed 16. October 2014)
8. ENISA Report on Digital Honeypots: Cyber security according to Winnie the Pooh: new report by EU Agency ENISA on 'digital trap' honeypots to detect cyber-attacks creates a buzz http://www.enisa.europa.eu/media/press-releases/new-report-by-eu-agency-enisa-on-digital-trap-honeypots-to-detect-cyber-attacks (accessed 16. October 2014)
9. Malicious DNS World Activity http://exposure.iseclab.org/ (accessed 16. October 2014)
10. Writing Snort Rules http://manual.snort.org/node27.html (accessed 16. October 2014)
11. Payload Detection http://manual.snort.org/node32.html (accessed 16. October 2014)
12. Scapy http://www.secdev.org/projects/scapy/ (accessed 16. October 2014)
13. Ramnit Goes Social http://www.seculert.com/blog/2012/01/ramnit-goes-social.html (accessed 16. October 2014)
14. Hex To ASCII Converter http://dolcevie.com/js/converter.html (accessed 16. October 2014)
15. Writing YARA rules https://github.com/plusvic/yara/blob/master/docs/writingrules.rst (accessed 16. October 2014)
16. YARA homepage http://plusvic.github.io/yara/ (accessed 16. October 2014)
17. YARA manual https://googledrive.com/host/0BznOMqZ9f3VUek8yN3VvSGdhRFU/YARA-Manual.pdf (accessed 16. October 2014)
18. OpenIOC home page http://www.openioc.net/ (accessed 16. October 2014)
19. OpenAppID Install Video (How to install snort with OpenAppID on a clean Ubuntu system) http://blog.snort.org/2014/03/openappid-install-video.html (accessed 16. October 2014)
20. SNORT Manual: Writing Snort Rules http://manual.snort.org/node27.html
21. PCRE – Perl Compatible Regular Expressions http://www.pcre.org/ (accessed 16. October 2014)
22. Writing Snort Rules Correctly http://blog.joelesler.net/2010/02/writing-snort-rules-correctly.html (accessed 16. October 2014)
23. Offset, Depth, Distance, and Within http://blog.joelesler.net/2010/03/offset-depth-distance-and-within.html (accessed 16. October 2014)
24. Analysing a Hack from A to Z http://www.windowsecurity.com/articles-tutorials/misc_network_security/Analyzing-Hack-Part1.html (accessed 16. October 2014)
25. An Easy Way to Test Your Snort Rules http://www.lteo.net/blog/2012/10/26/an-easy-way-to-test-your-snort-rules/ (accessed 16. October 2014)
26. rule2alert https://code.google.com/p/rule2alert/ (accessed 16. October 2014)

ENISA
European Union Agency for Network and Information Security
Science and Technology Park of Crete (ITE)
Vassilika Vouton, 700 13, Heraklion, Greece

Athens Office
1 Vass. Sofias & Meg. Alexandrou
Marousi 151 24, Athens, Greece

PO Box 1309, 710 01 Heraklion, Greece
Tel: +30 28 14 40 9710
info@enisa.europa.eu
www.enisa.europa.eu